Bit-Serial Logic Operation Processor

Introduction

In this lab, we are designing and building a bit-serial logic operation processor with the use of several logic gates, multiplexers, and a counter. The lab consists of two parts. In the first, we designed and built a 4-bit physical circuit. In the second, we extended the design on Vivado using SystemVerilog to an 8-bit processor and implemented the design on the Spartan-7 FPGA. The processor has four main units: register, computation, routing, and control. The register unit stores the loaded or computed result of inputs A and B. The computation unit can compute the result of the eight logic operations AND, OR, XOR, 1, NAND, NOR, XNOR, and 0. The result then goes to the routing unit, where you can control if the result should be stored in either A or B, keep A and B the same, or swap A with B. This whole process is controlled through the Finite State Machine (FSM) of the control unit.

Operation of the Logic Processor

To load a value in either A or B, you should first flip the 4 switches (D_3 - D_0) on the FPGA for the wanted value, then flip the switch LOAD A/B as shown in Fig.1. Then, to perform an operation, you want to set up the desired computation and route you wish for the result. The 3 switches $F_2F_1F_0$ decide which operation will be performed, and the switches R_1R_0 control the route. After setting the operation and route, you can hit EXECUTE, and the new result can be seen on the registers A and B.

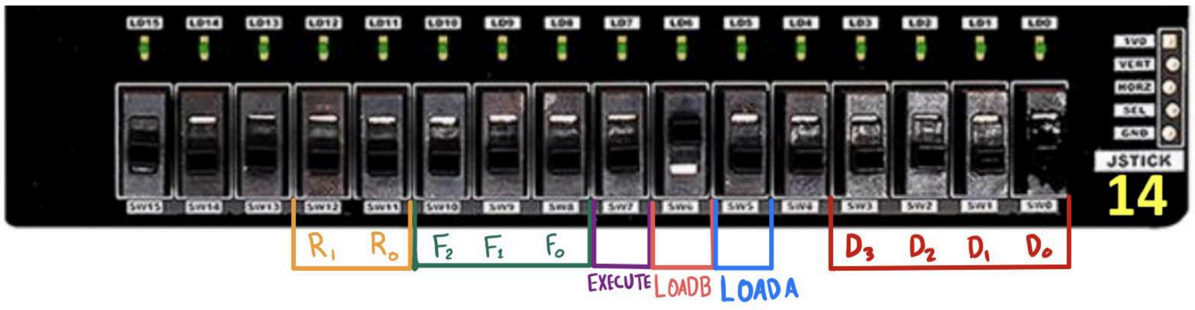
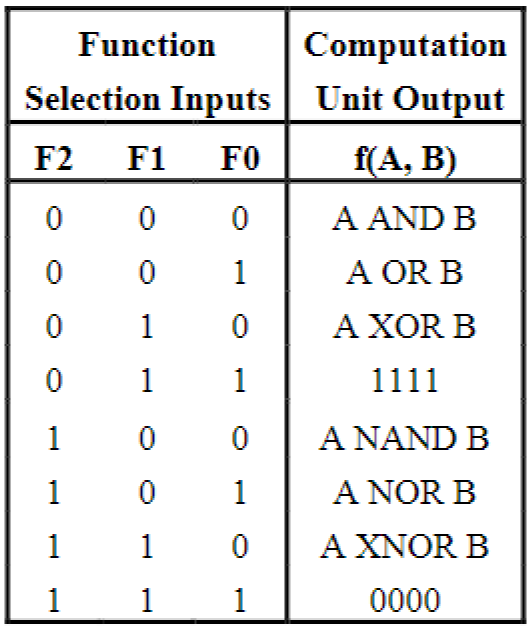
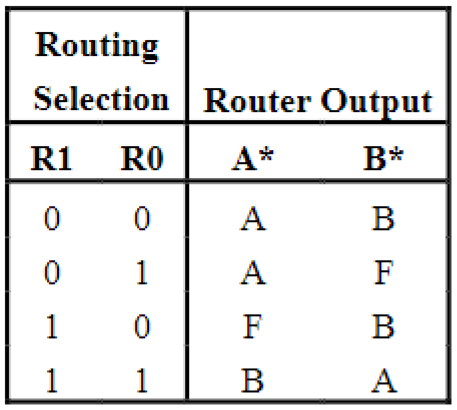


Fig.1: View of the circuit's input switches



Function Selection Inputs			Computation Unit Output
F2	F1	F0	f(A, B)
0	0	0	A AND B
0	0	1	A OR B
0	1	0	A XOR B
0	1	1	1111
1	0	0	A NAND B
1	0	1	A NOR B
1	1	0	A XNOR B
1	1	1	0000



Routing Selection		Router Output	
R1	R0	A*	B*
0	0	A	B
0	1	A	F
1	0	F	B
1	1	B	A

Table.1: Table for the F and R signals that control the computation and routing units

Description:

- Register Unit:

  The register unit consists of two 4-bit shift registers, RegA and RegB, which hold the values for A and B. The values of the registers are always displayed on the multisegment LEDs. The inputs of the registers are either loaded by the user or serially from the routing unit. The control unit's output S combined with Load A and Load B signals go through the combinational logic block, as shown in Fig.2, to control whether the registers should hold their value, shift, or load D3-D0 from switches. We can control this through S1 and S0; whenever S1S0 = 00, that's the hold state. S1S0 = 01, that's the shift state. S1S0 = 11, that is the load state. The expressions for both S1 and S0, alongside the truth table, can be found in Table.2 below.

LD A/B	S	S1	S0
0	0	0	0
0	1	0	1
1	0	1	1
1	1	0	1

Table.2: Truth table for S1 and S0 inputs of the register

S1 = LDA\B*S' **S0 = LDA/B+S**

- Computation Unit:

    The computation unit takes the 1-bit input from RegA and RegB and the select inputs $F_2F_1F_0$. As can be seen in Table. 1, the unit will output the function f(A, B), which is chosen in a multiplexer by the select inputs, and it will output inputs A and B unchanged.

- Routing Unit:

    The outputs f(A, B), A, and B from the computation unit will go through a multiplexer in the routing unit to choose the route in which it will be stored. This decision is based on the select inputs R_1R_0. As can be seen in Table.1, the result can be stored in either RegA or RegB while keeping the other input the same, or RegA and RegB keep their values in the original place or swapped.

- Control Unit:

    The control unit has inputs LOAD A, LOAD B, EXECUTE, and a clock signal as shown in Fig.2. The LOAD A and LOAD B are responsible for parallel loading the RegA and RegB, respectively. Similarly, the EXECUTE input is responsible for executing the selected operation on the stored inputs in RegA and RegB and sending the selected path to the routing unit to store the result. The clock is used for the registers and the FSM's flip-flop so that the computation cycle works in an instant manner, but enabling the user to debug and observe each step.

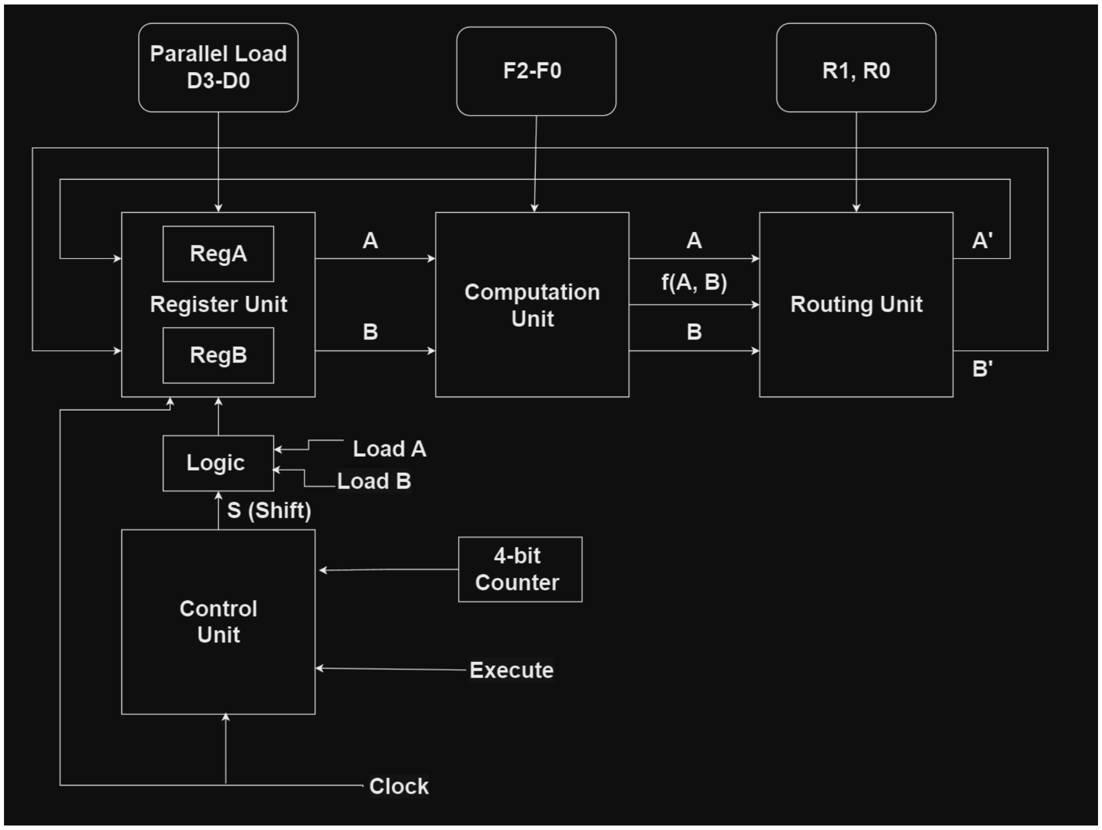


Fig.2: High-level block diagram for the 4-bit processor

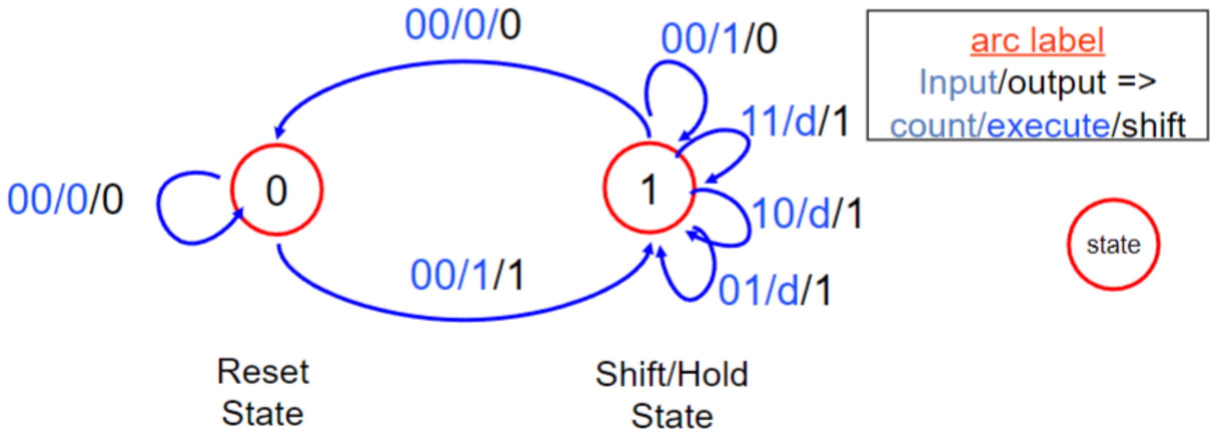


Fig.3: State transition diagram for the control unit FSM

➢ FSM:

The most important part of the control unit is the FSM, which controls the states of the whole processor. In our design, we used a Mealy machine, which has fewer states than its Moore counterpart. We had only two states, which can be described as the reset state and the shift/hold state, as shown in Figure 4. There are four signals that control the state transition, which are the input EXECUTE, a single-bit state representation Q, and the least two significant bits of a counter C1C0, as can be seen from Table.3 below. The output of this FSM is the signal S, which stands for shift and the new state representation Q^+, the expressions for both outputs can be found in Fig.3. The output S is what controls the registers RegA and RegB. Whenever S is 1, the registers will start to shift. Otherwise, it will depend on the signals LOAD A and LOAD B to load or just hold the value.

Exec. Switch ('E')	Q	C1	C0	Reg. Shift ('S')	Q^+	$C1^+$	$C0^+$
0	0	0	0	0	0	0	0
0	0	0	1	d	d	d	D
0	0	1	0	d	d	d	D
0	0	1	1	d	d	d	D
0	1	0	0	0	0	0	0
0	1	0	1	1	1	1	0
0	1	1	0	1	1	1	1
0	1	1	1	1	1	0	0
1	0	0	0	1	1	0	1
1	0	0	1	d	d	d	D
1	0	1	0	d	d	d	D
1	0	1	1	d	d	d	D
1	1	0	0	0	1	0	0
1	1	0	1	1	1	1	0
1	1	1	0	1	1	1	1
1	1	1	1	1	1	0	0

Table.3: Control unit state transition table using the Mealy state machine

Circuit Schematic

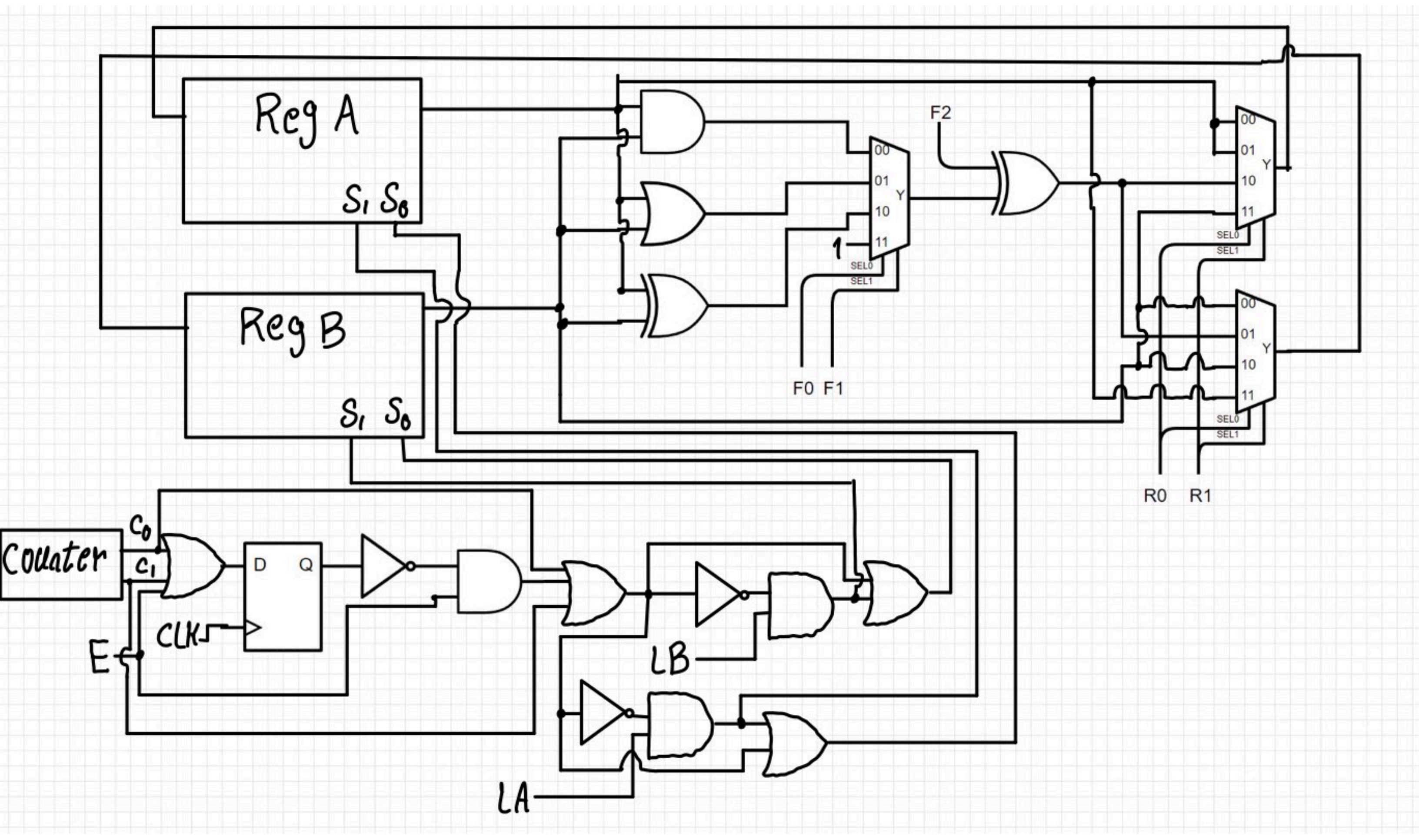


Fig.5: Circuit schematic for the 4-bit processor

RTL Block Diagram

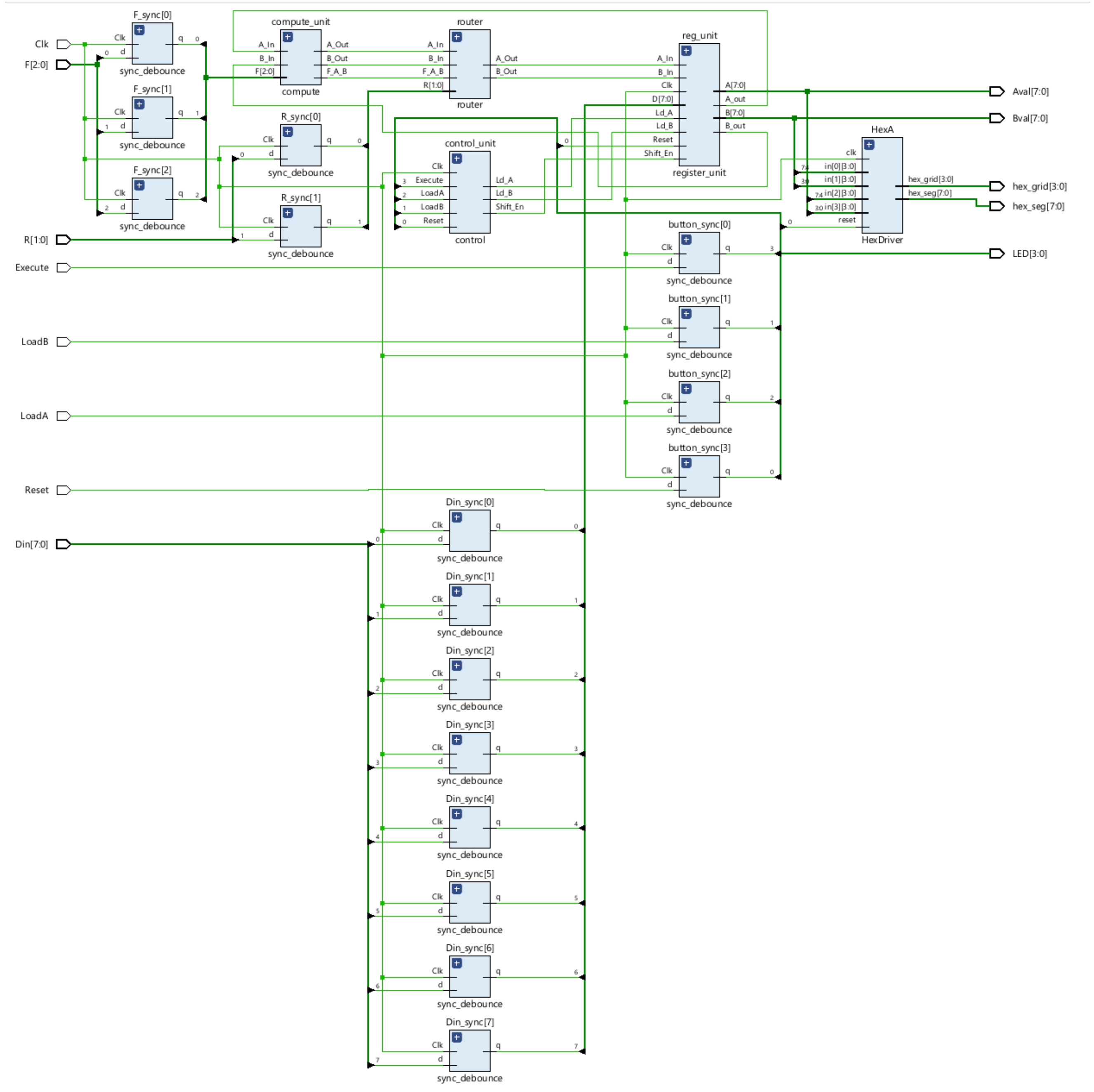


Fig.7: RTL block diagram for the 8-bit processor from Vivado

Simulation of The Processor

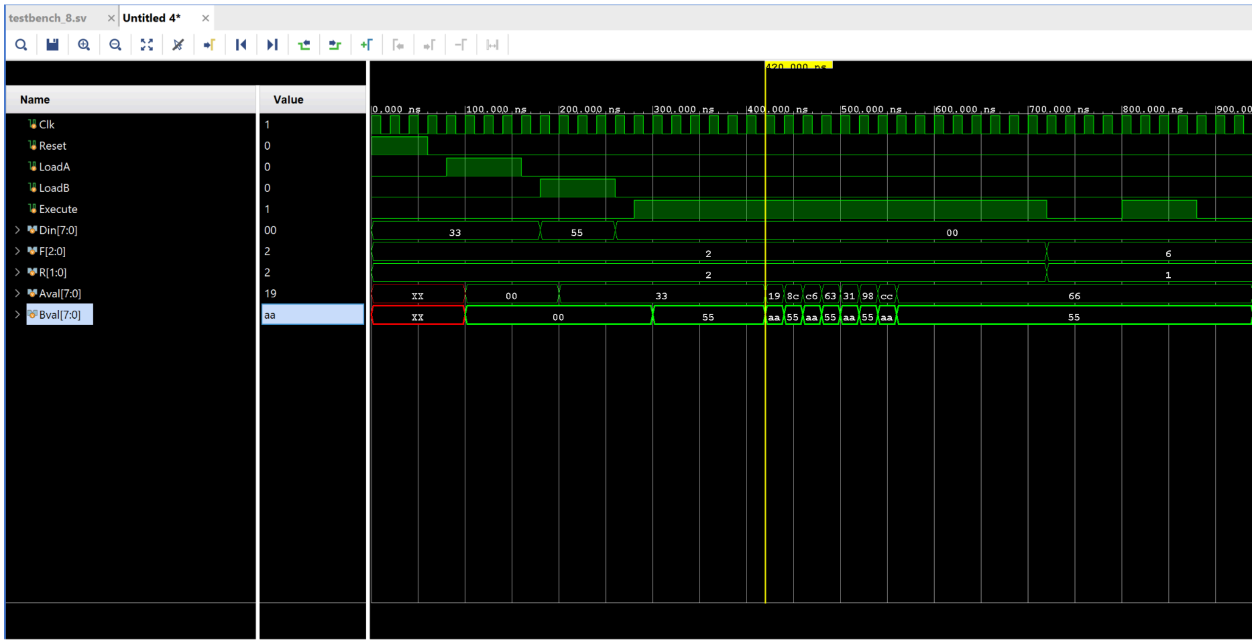


Fig.8: Behavioral Simulation of the 8-bit Processor

Fig.8 shows the behavioral simulation of the 8-bit processor with various signals. As seen in the waveform, when LoadA was pressed, Aval got loaded with Din (x33). Similarly, when LoadB was pressed, Bval got loaded with the updated value of Din (x55). We can also see that the F[2:0] signal is set to 2, meaning that the operation performed is XOR, as shown in Table.1. Moreover, the R[1:0] signal is set to 2, which means that the result of XORing A and B will be stored in register B; meanwhile A will remain the same. Additionally, we can observe the sequence of right shifts happening at 420ns, which is required to give us the correct result. In summary, this behavioral simulation shows the result of A XOR B and stores it in register B while preserving register A.

Conclusion

In this lab, we successfully designed and implemented a bit-serial logic operation processor through both physical circuit construction and an FPGA-based extension composed of 4 main units: a register unit, a computation unit, a routing unit, and a control unit. The processor was capable of parallel loading RegA and RegB with input D[3:0], performing eight basic logic operations AND, OR, XOR, 1, NAND, NOR, XNOR, and 0, and routing the computed results between two registers determined by the inputs F[2:0] and R[1:0]. The computation cycle was organized using a Mealy Finite State Machine (FSM). The two states of the FSM depended on the EXECUTE signal, the state representation Q, and the counter's 2 least significant bits C_1C_0. We built a 4-bit processor physically, using multiplexers, counters, registers, a flip-flop, and logic gates, and then extended this design to an 8-bit processor using SystemVerilog on Vivado, which required some level of understanding of the syntax. By utilizing a modular approach, we were able to ensure flexibility in both the design and debugging phases, allowing for easier troubleshooting.